

# Fast Algorithms for the Maximum Clique Problem on Massive Sparse Graphs

Bharath Pattabiraman<sup>1</sup>, Md. Mostofa Ali Patwary<sup>1</sup>,  
Assefaw H. Gebremedhin<sup>2</sup>, Wei-keng Liao<sup>1</sup>, and Alok Choudhary<sup>1</sup>

<sup>1</sup> Northwestern University, Evanston, IL.  
{bpa342,mpatwary,wkliao,choudhar}@eecs.northwestern.edu

<sup>2</sup> Purdue University, West Lafayette, IN.  
agebrema@purdue.edu

**Abstract.** The maximum clique problem is a well known NP-Hard problem with applications in data mining, network analysis, informatics, and many other areas. Although there exist several algorithms with acceptable runtimes for certain classes of graphs, none of them are feasible for massive graphs. We present a new exact algorithm which employs novel pruning techniques to very quickly solve for the maximum clique on large sparse graphs. Extensive experiments on several types of synthetic and real-world graphs show that our new algorithm is up to several orders of magnitude faster than existing algorithms for most instances. We also present a heuristic variant that runs orders of magnitude faster than the exact algorithm, while providing optimal or near-optimal solutions. The new algorithms, unlike existing algorithms, are also well suited for parallelization.

**Keywords:** maximum clique problem, DIMACS Challenge, branch-and-bound algorithms, pruning, sparse graphs.

## 1 Introduction

A clique in an undirected graph is a subset of vertices in which every two vertices are adjacent to each other. The *maximum* clique problem is to find the clique of largest possible size in a given graph.

The maximum clique problem, and the related *maximal* clique and clique *enumeration* problems, find applications in many areas. Some examples include data mining [10], information retrieval [1], social networks [12], bioinformatics [17], computer vision [14], coding [5], and economics [2]. An example of its application can be given using data mining or information retrieval, where one needs to retrieve data that are considered similar based on some given metric. A graph is constructed with vertices corresponding to data items and edges connecting similar vertices. Finding a clique in such a graph gives a cluster of similar data. Such problems also arise in a number of other areas including identification and classification of new diseases based on symptom correlation [4], pattern recognition [22], and bioinformatics [17]. More recently, the maximum clique problem has seen important applications in social network analysis, primarily in community detection [12,20,23]. More examples of application areas for clique problems can be found in [21].

The maximum clique problem is NP-Hard [13]. Most exact algorithms for solving it employ some form of *branch-and-bound* approach. While branching systematically searches for all candidate solutions, bounding (also known as *pruning*) discards fruitless candidates based on a previously computed bound. An early example of a simple and effective branch-and-bound algorithm for the maximum clique problem is one by Carraghan and Pardalos [6]. More recently, Östergård [19] introduced an improved algorithm and demonstrated its relative advantages via computational experiments. Other examples of *branch-and-bound* algorithms for the clique problem include [3,24,25].

The algorithms of [6] and [19] as well as many other existing branch-and-bound algorithms are feasible only for graphs with up to a few thousand vertices and a corresponding number of edges. In addition, most existing algorithms are inherently sequential. The ease with which an algorithm can be parallelized is important for handling large-scale graphs in emerging applications, where graphs with millions (or more) vertices are quite common [16].

In this paper, we present a new exact branch-and-bound algorithm for the maximum clique problem that employs several new pruning strategies, making it suitable for massive graphs. We show experimental results comparing the performance of our algorithm with the algorithm of Carraghan and Pardalos [6] and the algorithm of Östergård [19]. Our testbed includes DIMACS benchmark graphs [15], real-world graphs from various application domains, and large scale synthetic graphs of various structures. The new algorithm is found to be up to orders of magnitude faster on large, sparse graphs and of comparable runtime on denser graphs. We also present a heuristic based on similar pruning techniques as the exact algorithm. The heuristic performs several orders of magnitude faster than the exact algorithm, while delivering solutions that are optimal or near-optimal for most cases. Both the heuristic and the exact algorithm are well-suited for parallelization.

The remainder of this paper is organized as follows. We provide a brief summary of related work in Section 2. In Section 3, we propose our algorithms. We present our experimental methodology and results in Section 4. We conclude our work and propose future lines of research in Section 5.

## 2 Related Work

Given a simple undirected graph  $G$ , the maximum clique can clearly be obtained by enumerating *all* of the cliques present in it and picking the largest of them. Carraghan and Pardalos [6] introduced a simple-to-implement algorithm that avoids enumerating all cliques and instead works with a significantly reduced partial enumeration. The reduction in enumeration is achieved via a *pruning* strategy which reduces the search space tremendously. The algorithm works by performing at each step  $i$ , a *depth first search* from vertex  $v_i$ , where the goal is to find the largest clique containing the vertex  $v_i$ . At each *depth* of the search, the algorithm compares the number of remaining vertices that could potentially constitute a clique containing vertex  $v_i$  against the size of the largest clique encountered thus far. And if that number is found to be smaller, the algorithm backtracks (the search is pruned).

Östergård [19] devised an algorithm that incorporated an additional pruning strategy to the one by Carraghan and Pardalos. The opportunity for the new pruning strategy is created by *reversing* the order in which the search is done by the Carraghan-Pardalos algorithm. This allows for an additional pruning with the help of some auxiliary bookkeeping. Experimental results in [19] showed that the Östergård algorithm is faster than the one by Carraghan-Pardalos on random and DIMACS benchmark graphs [15]. However, the new pruning strategy used in this algorithm is intimately tied to the order in which vertices are processed, introducing an inherent sequentiality into the algorithm which renders it unsuitable for easy parallelization.

A number of other algorithms also use approximate vertex-coloring to obtain an upper bound on the maximum clique [24,25]. *Vertex-coloring* refers to coloring each vertex in a given graph such that pairwise adjacent vertices achieves different colors. The number of colors used gives an upper bound on the maximum clique of the graph, which is then used to reduce the search space.

### 3 New Algorithms

While the Carraghan-Pardalos algorithm is not the fastest, the one by Östergård is inherently sequential, and more recent ones have a complex implementation. Our new algorithms consolidate all their good features - they are very fast, simple to implement and do not require any auxiliary bookkeeping. And moreover, they are easy to parallelize as they do *not* rely on the order in which the vertices are processed.

In the following subsections, we present our new exact algorithm and heuristic. Before going into the details of the algorithms, we introduce a few notations used throughout the paper. We identify the  $n$  vertices of the input graph  $G = (V, E)$  as  $\{v_1, v_2, \dots, v_n\}$ . The set of vertices adjacent to a vertex  $v_i$ , the set of its neighbors, is denoted by  $N(v_i)$ . And the cardinality of  $N(v_i)$ , its degree, is denoted by  $d(v_i)$ .

#### 3.1 The Exact Algorithm

The maximum clique in a graph can be found by computing the largest clique containing each vertex and picking the largest among these. A key element of our exact algorithm is that during the search for the largest clique containing a given vertex, vertices that cannot form cliques larger than the current maximum clique are *pruned*, in a hierarchical fashion. The method is outlined in detail in Algorithm 1. Throughout the algorithm, the variable *max* stores the size of the maximum clique found thus far. Initially it is set to be equal to the lower bound *lb* provided as an input parameter, and it gives the maximum clique size when the algorithm terminates.

To obtain the largest clique containing a vertex  $v_i$ , it is sufficient to consider only the neighbors of  $v_i$ . The main routine MAX-CLIQUE thus generates for each vertex  $v_i \in V$  a set  $U \subseteq N(v_i)$  (neighbors of  $v_i$  that survive pruning) and calls the subroutine CLIQUE on  $U$ . The subroutine CLIQUE goes through every relevant clique containing  $v_i$  in a recursive fashion and returns the largest. The subroutine is similar to the Carraghan-Pardalos algorithm [6]. We use *size* to maintain the size of the clique found at any point through the recursion. Since we start with a clique of just one vertex, the value of *size*

is set to be one initially when the subroutine `CLIQUE` is called (Line 10 of Algorithm 1).

Our algorithm consists of several pruning steps. The pruning in Line 4 of `MAX-CLIQUE` filters vertices having strictly fewer neighbors than the size of the maximum clique already computed. These vertices can be safely ignored, since even if a clique were to be found, its size would not be larger than  $max$ . While forming the neighbor list  $U$  for a vertex  $v_i$ , we include only those of  $v_i$ 's neighbors for which the largest clique

---

**Algorithm 1** Algorithm for finding the maximum clique of a given graph. *Input:* Graph  $G = (V, E)$ , lower bound on clique  $lb$  (default, 0). *Output:* Size of maximum clique.

---

```

1: procedure MAXCLIQUE( $G = (V, E), lb$ )
2:    $max = lb$ 
3:   for  $i : 1$  to  $n$  do
4:     if  $d(v_i) \geq max$  then                                ▷ Pruning 1
5:        $U \leftarrow \emptyset$ 
6:       for each  $v_j \in N(v_i)$  do
7:         if  $j > i$  then                                    ▷ Pruning 2
8:           if  $d(v_j) \geq max$  then                            ▷ Pruning 3
9:              $U \leftarrow U \cup \{v_j\}$ 
10:      CLIQUE( $G, U, 1$ )

```

---

– Subroutine

```

1: procedure CLIQUE( $G = (V, E), U, size$ )
2:   if  $U = \emptyset$  then
3:     if  $size > max$  then
4:        $max = size$ 
5:     return
6:   while  $|U| > 0$  do
7:     if  $size + |U| \leq max$  then                                ▷ Pruning 4
8:       return
9:     Select any vertex  $u$  from  $U$ 
10:     $U \leftarrow U \setminus \{u\}$ 
11:     $N'(u) := \{w | w \in N(u) \wedge d(w) > max\}$  ▷ Pruning 5
12:    CLIQUE( $G, U \cap N'(u), size + 1$ )

```

---

containing them has not been found (Line 7), to avoid recomputing previously found cliques. Furthermore, the pruning in Line 8 excludes vertices  $v_j \in N(v_i)$  that have degree less than the current value of  $max$ , since any such vertex could not form a clique of size larger than  $max$ . The pruning strategy in Line 7 of subroutine `CLIQUE` checks for the case where even if all vertices of  $U$  were added to get a clique, its size would not exceed that of the largest clique encountered so far in the search,  $max$ . Line 11 of `CLIQUE` reduces the number of comparisons needed to generate the intersection set in line 12.

The pruning in Line 7 of the subroutine has been used in existing algorithms [6] and [19], whereas the other pruning steps have been introduced by us. As shall be seen later, the newer pruning steps turn out to be very effective for large sparse graphs.

The algorithm, as presented, finds the size of the maximum clique. Finding the actual set of vertices that form the clique is a trivial extension that we ignore for simplicity.

### 3.2 The Heuristic

The exact algorithm examines for every vertex  $v_i$  all relevant cliques containing the vertex  $v_i$  in order to determine the clique of maximum size among them. Our heuristic speeds up this process by instead examining only a subset of the relevant cliques.

The heuristic is presented in Algorithm 2. The main routine is very similar to the main routine in Algorithm 1. The subroutine CLIQUEHEU considers only the *maximum degree* neighbor at each step instead of recursively considering all neighbors from the set  $U$ . Since we are looking for the largest clique containing each vertex, the maximum degree vertex is more likely to be a member of the largest clique com-

pared to the other vertices. The effect of choosing the maximum degree vertex as opposed to any random vertex will be analyzed in a later section. The pruning steps in Algorithm 2, are otherwise the same as in Algorithm 1, unless not applicable (Pruning 2 and 4). It is to be noted that Turner [26] uses an algorithm similar in spirit to the subroutine of Algorithm 2 in his coloring algorithm.

### 3.3 Complexity

The exact algorithm, Algorithm 1, examines for every vertex  $v_i$  all candidate cliques containing the vertex  $v_i$  in its search for the largest clique. Its time complexity is exponential in the worst case. The heuristic, Algorithm 2, loops over the  $n$  vertices, each time possibly calling the subroutine CLIQUEHEU, which effectively is a loop that runs until the set  $U$  is empty. Clearly,  $|U|$  is bounded by the max degree  $\Delta$  in the graph. The subroutine also includes the computation of a neighbor list, whose runtime is bounded by  $O(\Delta)$ . Thus, the time complexity of the heuristic is  $O(n \cdot \Delta^2)$ .

### 3.4 Parallelization

In our algorithms, there is no dependency or strict order in which the vertices have to be computed unlike existing algorithms. The sizes of the largest cliques containing different vertices can be computed mostly independently, and thus can be done in parallel by different computational units.

## 4 Experiments and Results

We present in this section experimental results comparing the performance of our algorithm with the Carraghan-Pardalos algorithm [6] and the Östergård algorithm [19]. We

---

**Algorithm 2** Heuristic for finding the maximum clique in a graph. *Input:* Graph  $G = (V, E)$ . *Output:* Approximate size of maximum clique.

---

```

1: procedure MAXCLIQUEHEU( $G = (V, E)$ )
2:   for  $i : 1$  to  $n$  do
3:     if  $d(v_i) \geq \text{max}$  then                                ▷ Pruning 1
4:        $U \leftarrow \emptyset$ 
5:       for each  $v_j \in N(v_i)$  do
6:         if  $d(v_j) \geq \text{max}$  then                                ▷ Pruning 3
7:            $U \leftarrow U \cup \{v_j\}$ 
8:       CLIQUEHEU( $G, U, 1$ )

```

---

– Subroutine

```

1: procedure CLIQUEHEU( $G = (V, E), U, \text{size}$ )
2:   if  $U = \emptyset$  then
3:     if  $\text{size} > \text{max}$  then
4:        $\text{max} = \text{size}$ 
5:     return
6:   Select a vertex  $u \in U$  of maximum degree in  $G$ 
7:    $U \leftarrow U \setminus \{u\}$ 
8:    $N'(u) := \{w | w \in N(u) \wedge d(w) > \text{max}\}$   ▷ Pruning 5
9:   CLIQUEHEU( $G, U \cap N'(u), \text{size} + 1$ )

```

---

**Table 1.** Structural properties (the number of vertices,  $|V|$ ; edges,  $|E|$ ; and the maximum degree,  $\Delta$ ) of the graphs,  $G$  in the testbed: DIMACS Challenge graphs (upper left); UF Collection (lower and middle left); RMat graphs (right).

$G$	$ V $	$ E $	$\Delta$	$G$	$ V $	$ E $	$\Delta$
<i>hamming6-4</i>	64	704	22	<i>rmat_er_1</i>	131,072	1,048,515	82
<i>johnson8-4-4</i>	70	1,855	53	<i>rmat_er_2</i>	262,144	2,097,104	98
<i>keller4</i>	171	9,435	124	<i>rmat_er_3</i>	524,288	4,194,254	94
<i>c-fat200-5</i>	200	8,473	86	<i>rmat_er_4</i>	1,048,576	8,388,540	97
<i>brock200-2</i>	200	9,876	114	<i>rmat_er_5</i>	2,097,152	16,777,139	102
<i>cond-mat-2003</i>	31,163	120,029	202	<i>rmat_sd1_1</i>	131,072	1,046,384	407
<i>email-Enron</i>	36,692	183,831	1,383	<i>rmat_sd1_2</i>	262,144	2,093,552	558
<i>dictionary28</i>	52,652	89,038	38	<i>rmat_sd1_3</i>	524,288	4,190,376	618
<i>Fault_639</i>	638,802	13,987,881	317	<i>rmat_sd1_4</i>	1,048,576	8,382,821	802
<i>audikw_1</i>	943,695	38,354,076	344	<i>rmat_sd1_5</i>	2,097,152	16,767,728	1,069
<i>bone010</i>	986,703	35,339,811	80	<i>rmat_sd2_1</i>	131,072	1,032,634	2,980
<i>af_shell10</i>	1,508,065	25,582,130	34	<i>rmat_sd2_2</i>	262,144	2,067,860	4,493
<i>as-Skitter</i>	1,696,415	11,095,298	35,455	<i>rmat_sd2_3</i>	524,288	4,153,043	6,342
<i>roadNet-CA</i>	1,971,281	2,766,607	12	<i>rmat_sd2_4</i>	1,048,576	8,318,004	9,453
<i>kkt_power</i>	2,063,494	6,482,320	95	<i>rmat_sd2_5</i>	2,097,152	16,645,183	14,066

implemented the algorithm of [6] ourselves, whereas for the algorithm of [19], we used the publicly available *cliquer* source code [18].

All our experiments are performed on a Linux workstation running 64-bit version Red Hat Enterprise Linux Server release 6.2, with a 2.00 GHz Intel Xeon E7540 processor. Our implementations are all in C++, and the codes are compiled using gcc version 4.4.6 with -O3 optimization.

Our testbed is grouped in five categories of graphs:

- **DIMACS** (5 graphs) - select graphs from the Second DIMACS Implementation Challenge [15].
- **Real-world applications** (10 graphs) - graphs downloaded from the University of Florida Sparse Matrix Collection [8]. The graphs represent various areas, including social computing, physics, linear programming, medical science, structural engineering, civil engineering, electrical engineering and automotive industry,.
- **Synthetic, Random** (5 graphs) - Erdős-Renyi random graphs generated using the R-MAT algorithm [7] with the parameters (0.25, 0.25, 0.25, 0.25). (The graphs are denoted with prefix *rmat\_er*.)
- **Synthetic, Skewed Degree, Type 1** (5 graphs) - graphs generated using R-MAT with the parameters (0.45, 0.15, 0.15, 0.25). (Denoted with prefix *rmat\_sd1*.)
- **Synthetic, Skewed Degree, Type 2** (5 graphs) - graphs generated using R-MAT with the parameters (0.55, 0.15, 0.15, 0.15). (Denoted with prefix *rmat\_sd2*.)

The DIMACS graphs are an established benchmark for the maximum clique problem, but they are of rather limited size and variation. We included the application (University of Florida) graphs and the synthetic (RMat) graphs to represent a wide spectrum of large graphs posing varying degrees of difficulty for testing the algorithms. The

**Table 2.** Comparison of runtimes (in seconds) of algorithms [6] (*CP*) and [19] (*cliquer*) with the time taken by our new exact algorithm ( $\tau_{new-exa}$ ) for the graphs in the testbed, with the fastest (marked in bold) for each case. An asterisk (\*) indicates that the algorithm did not terminate within 25,000 seconds for that instance.  $P1$ ,  $P2$ ,  $P3$  and  $P5$  are the number of vertices pruned in steps Pruning 1, 2, 3 and 5 of Algorithm 1.  $\mu$  denotes the maximum clique size,  $\mu_{new-heu}$ , the maximum clique size returned by our heuristic and  $\tau_{new-heu}$ , its runtime. For the graph *rmat\_sd2\_5*, none of the algorithms computed the maximum clique size in a reasonable time; the entry is marked *N*, denoting “Not Known”.

$G$	$\mu$	$\tau_{CP}$	$\tau_{cliquer}$	$\tau_{new-exa}$	$P1$	$P2$	$P3$	$P5$	$\mu_{new-heu}$	$\tau_{new-heu}$
<i>hamming6-4</i>	4	<0.01	<0.01	<0.01	0	704	0	0	4	<0.01
<i>johnson8-4-4</i>	14	0.19	<0.01	0.23	0	1855	0	0	14	<0.01
<i>keller4</i>	11	22.19	<b>0.15</b>	23.35	0	9435	0	0	11	<0.01
<i>c-fat200-5</i>	58	0.60	<b>0.33</b>	0.93	0	8473	0	0	58	0.04
<i>brock200-2</i>	12	0.98	<b>0.02</b>	1.10	0	9876	0	0	10	<0.01
<i>cond-mat-2003</i>	25	4.875	11.17	<b>0.011</b>	29,407	48,096	6,527	17,576	25	<0.01
<i>email-Enron</i>	20	7.005	15.08	<b>0.998</b>	32,462	155,344	4,060	8,835,739	18	0.261
<i>dictionary28</i>	26	7.700	32.74	< <b>0.01</b>	52,139	4,353	2,114	107	26	<0.01
<i>Fault_639</i>	18	14571.20	4437.14	<b>20.03</b>	36	13,987,719	126	1,116	18	5.80
<i>audikw_1</i>	36	*	9282.49	<b>190.17</b>	4,101	38,287,830	59,985	721,938	36	58.38
<i>bone010</i>	24	*	10002.67	<b>393.11</b>	37,887	34,934,616	361,170	43,991,787	24	24.39
<i>af_shell10</i>	15	*	21669.96	<b>50.99</b>	19	25,582,015	75	2,105	15	10.67
<i>as-Skitter</i>	67	24385.73	*	<b>3838.36</b>	1,656,570	6,880,534	981,810	737,899,486	66	27.08
<i>roadNet-CA</i>	4	*	*	<b>0.05</b>	1,487,640	1,079,025	370,206	4,302	4	0.08
<i>kkt_power</i>	11	*	*	<b>0.28</b>	1,166,311	4,510,661	401,129	1,978,595	11	0.48
<i>rmat_er_1</i>	3	256.37	215.18	<b>0.38</b>	780	1,047,599	915	8,722	3	0.12
<i>rmat_er_2</i>	3	1016.70	865.18	<b>0.78</b>	2,019	2,094,751	2,351	23,908	3	0.24
<i>rmat_er_3</i>	3	4117.35	3456.39	<b>1.87</b>	4,349	4,189,290	4,960	50,741	3	0.49
<i>rmat_er_4</i>	3	16419.80	13894.52	<b>4.16</b>	9,032	8,378,261	10,271	106,200	3	1.44
<i>rmat_er_5</i>	3	*	*	<b>9.87</b>	18,155	16,756,493	20,622	212,838	3	2.57
<i>rmat_sd1_1</i>	6	225.93	214.99	<b>1.39</b>	39,281	1,004,660	23,898	542,245	6	0.45
<i>rmat_sd1_2</i>	6	912.44	858.80	<b>3.79</b>	90,010	2,004,059	56,665	1,399,314	6	0.98
<i>rmat_sd1_3</i>	6	3676.14	3446.02	<b>8.17</b>	176,583	4,013,151	106,543	2,677,437	6	1.78
<i>rmat_sd1_4</i>	6	14650.40	13923.93	<b>25.61</b>	369,818	8,023,358	214,981	5,566,602	6	4.05
<i>rmat_sd1_5</i>	6	*	*	<b>46.89</b>	777,052	16,025,729	455,473	12,168,698	6	9.39
<i>rmat_sd2_1</i>	26	427.41	<b>213.23</b>	242.20	110,951	853,116	88,424	614,813,037	26	32.83
<i>rmat_sd2_2</i>	35	4663.62	<b>851.84</b>	3936.55	232,352	1,645,086	195,427	1,044,068,886	35	95.89
<i>rmat_sd2_3</i>	39	13626.23	<b>3411.14</b>	10647.84	470,302	3,257,233	405,856	1,343,563,239	37	245.51
<i>rmat_sd2_4</i>	43	*	<b>13709.52</b>	*	*	*	*	*	42	700.05
<i>rmat_sd2_5</i>	N	*	*	*	*	*	*	*	51	1983.21

*rmat\_er* graphs have *normal* degree distribution, whereas the *rmat\_sd1* and *rmat\_sd2* graphs have skewed degree distributions and contain many dense local subgraphs. The *rmat\_sd1* and *rmat\_sd2* graphs differ primarily in the magnitude of maximum vertex degree they contain; the *rmat\_sd2* graphs have much higher maximum degree. Table 1 provides structural information of all 30 test graphs.

Table 2 shows performance results of our exact algorithm and the algorithms of Caraghan and Pardalos [6] (*CP*) and Östergård [19] (*cliquer*) for the graphs in the testbed. It also shows the number of vertices discarded by pruning steps, Pruning 1, 2, 3, and 5 of Algorithm 1. Vertices that are discarded by Pruning 1 are skipped in the main loop of the algorithm, and the largest cliques containing them are not computed. The effect of Pruning 2 is to prevent inclusion of the previously computed  $k - 1$  vertices from the initial neighbor list of the vertex  $v_k$ . In the absence of Pruning 1, the number of vertices pruned by this step would be  $|E|$ . While Pruning 3 reduces the size of the input set on which the maximum clique is to be computed, Pruning 5 brings down the time taken to generate the intersection set in Line 12 of the subroutine. Since Pruning 4

is used in the other two algorithms as well, we do not report its impact. Since our new pruning steps hierarchically reduce the search space, one can expect a decreasing order of effect in the performance of Prunings 1, 2, 3 and 5, with Pruning 1 having the highest impact, and 5 the lowest. The last two columns also show the results of our heuristic i.e. the size of the maximum clique returned and runtime. Below, we analyze how our algorithms perform on different types of graphs in our testbed.

*Exact algorithms* As expected, our exact algorithm gives the same size of maximum clique as algorithms CP and *cliquer* for all test cases. Following is an analysis of the runtimes of these algorithms on the various categories of graphs in the testbed.

**a) DIMACS Graphs.** The runtime of our exact algorithm for the DIMACS graphs is in most cases comparable to that of CP and higher than that of *cliquer*. As can be seen from the Table 2, only Pruning 2 is effective for these graphs, and thus the performance results agree with one’s expectation. We include in the Appendix timing results on a larger collection of DIMACS graphs.

It is to be noted that the DIMACS graphs are intended to serve as challenging test cases for the maximum clique problem, and graphs with such high edge densities and low vertex count are rather rare in practice. For example, most of them have between 20 to 1024 vertices with an average edge density of  $\sim 0.6$ . However, most real world graphs are often very large and sparse. Good examples are Internet topology graphs [11], the web graph [16], social network graphs [9], and the real-world graphs in our testbed.

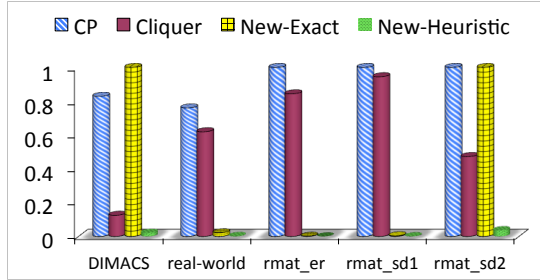
**b) Real-world Graphs.** For most of the graphs in this category, it can be seen that our algorithm runs several orders of magnitude faster than the other two, reflecting the large amount of pruning our algorithm has enforced. These numbers also illustrate the great benefit of hierarchical pruning. For the graphs *Fault\_639* and *af\_shell10*, there is only very minimal impact by Prunings 1, 3 and 5, however, in these cases Pruning 2 steps in to make a huge impact, thus delivering an impressive runtime.

**c) Synthetic Graphs.** For the synthetic graph types *rmat\_er* and *rmat\_sd1*, our algorithm clearly outperforms the other two by a few orders of magnitude in all cases. This is primary due to the high number of vertices discarded by the new pruning steps. In particular, for *rmat\_sd1* graphs, between 30 to 37% of the vertices are pruned just in the step Pruning 1. For the *rmat\_sd2* graphs, which have relatively larger maximum clique and higher maximum degree than the *rmat\_sd1* graphs, our algorithm is faster than CP but slower than *cliquer*.

*The Heuristic* It can be seen that our heuristic runs several orders of magnitude faster than our exact algorithm, while delivering either a optimal or very close to the optimal solution. It gives the optimal solutions on 25 out of the 30 test cases. On the rest 5 cases where it was suboptimal, it’s accuracy ranges from 83% to 99% (on average 93%). Additionally, we run the heuristic by choosing a vertex randomly in Line 6 of Algorithm 2 instead of the one with the maximum degree. We observe that on average, the solution is optimal only for  $< 40\%$  of the test cases compared to 83% when selecting the maximum degree vertex.



Figure 1 provides an aggregated visual summary of the runtime trends of the various algorithms across the five categories of graphs in the testbed.



**Fig. 1.** Runtime (normalized, mean) comparison between various algorithms. For each category of graph, first, all runtimes for each graph were normalized by the runtime of the slowest algorithm for that graph, and then the mean was calculated for each algorithm. Graphs were considered only if runtimes for at least three algorithms were available.

## 5 Conclusion and Future Work

We presented a new exact and a new heuristic algorithm for the maximum clique problem. We performed extensive experiments on five categories of graphs comparing the performance of our algorithms to the ones by Carraghan and Pardalos (CP) [6] and Östergård (*cliquer*) [19]. For DIMACS benchmark graphs and certain dense synthetic graphs (*rmat\_sd2*), our new exact algorithm performs comparably with the CP algorithm, but slower than *cliquer*. For large sparse graphs, both synthetic and real-world, our new algorithm runs several orders of magnitude faster than the other two. The heuristic, which runs many orders of magnitude faster than our exact algorithm and the others, gives optimal solution for 83% of the test cases, and when it is sub-optimal, its accuracy ranges between 0.83 and 0.99.

Although our algorithms work with any vertex ordering, it would be interesting to study the variation of its performance with the ordering employed. We intend to conduct a thorough analysis of the effect of different ordering techniques on our algorithms in the future. We also plan to study the merit of a hybrid approach - using the size of the maximum clique obtained by the heuristic as a lower bound to the exact algorithm which could improve pruning. It would also be of interest to compare our algorithms with other more recent ones such as [24,25]. Due to complexity of their implementation, and unavailability of source codes, we were unable to include it in this work. Since the presented algorithms are well suited for parallelization, implementation of parallel versions and an analysis of their performance on different computing platforms on very massive instances are also interesting lines of future research.

## References

1. J. G. AUGUSTSON AND J. MINKER, *An analysis of some graph theoretical cluster techniques*, J. ACM, 17 (1970), pp. 571–588.
2. V. BOGINSKI, S. BUTENKO, AND P. M. PARDALOS, *Statistical analysis of financial networks*, Computational Statistics & Data Analysis, 48 (2005), pp. 431–443.
3. I. M. BOMZE, M. BUDINICH, P. M. PARDALOS, AND M. PELILLO, *The maximum clique problem*, in Handbook of Combinatorial Optimization, Kluwer Academic Publishers, 1999, pp. 1–74.

4. R. E. BONNER, *On some clustering techniques*, IBM J. Res. Dev., 8 (1964), pp. 22–32.
5. A. E. BROUWER, J. B. SHEARER, N. J. A. SLOANE, AND W. D. SMITH, *A new table of constant weight codes.*, IEEE Transactions on Information Theory, (1990), pp. 1334–1380.
6. R. CARRAGHAN AND P. PARDALOS, *An exact algorithm for the maximum clique problem*, Oper. Res. Lett., 9 (1990), pp. 375–382.
7. D. CHAKRABARTI AND C. FALOUTSOS, *Graph mining: Laws, generators, and algorithms*, ACM Comput. Surv., 38 (2006).
8. T. A. DAVIS, *The university of florida sparse matrix collection*, NA DIGEST, (1997).
9. P. DOMINGOS AND M. RICHARDSON, *Mining the network value of customers*, in Proc. of the 7th ACM SIGKDD KDD'01, KDD '01, New York, NY, USA, 2001, ACM, pp. 57–66.
10. J. EDACHERY, A. SEN, F. J. BRANDENBURG, F. J. BR., AND L. F. INFORMATIK, *Graph clustering using distance-k cliques*, in Proc. of Graph Drawing, Springer-Verlag, 1999, pp. 98–106.
11. M. FALOUTSOS, P. FALOUTSOS, AND C. FALOUTSOS, *On power-law relationships of the internet topology*, in Proc. of the conference on Applications, technologies, architectures, and protocols for computer communication, SIGCOMM '99, ACM, 1999, pp. 251–262.
12. S. FORTUNATO, *Community detection in graphs*, Physics Reports, 486 (2010), pp. 75–174.
13. M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1979.
14. R. HORAUD AND T. SKORDAS, *Stereo correspondence through feature grouping and maximal cliques*, IEEE Trans. Pattern Anal. Mach. Intell., 11 (1989), pp. 1168–1180.
15. D. JOHNSON AND E. M. A. TRICK, *Cliques, coloring and satisfiability: Second dimacs implementation challenge*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 26 (1996).
16. R. KUMAR, P. RAGHAVAN, S. RAJAGOPALAN, AND A. TOMKINS, *Extracting large-scale knowledge bases from the web.*, in VLDB'99, 1999, pp. 639–650.
17. T. MATSUNAGA, C. YONEMORI, E. TOMITA, AND M. MURAMATSU, *Clique-based data mining for related genes in a biomedical database*, BMC Bioinformatics, 10 (2009), p. 205.
18. S. NISKANEN AND P. R. J. ÖSTERGÅRD, *Cliquer user's guide, version 1.0*, Tech. Rep. T48, Communications Laboratory, Helsinki University of Technology, Espoo, Finland, 2003.
19. P. R. J. ÖSTERGÅRD, *A fast algorithm for the maximum clique problem*, Discrete Appl. Math., 120 (2002), pp. 197–207.
20. G. PALLA, I. DERENYI, I. FARKAS, AND T. VICSEK, *Uncovering the overlapping community structure of complex networks in nature and society*, Nature, 435 (2005), pp. 814–818.
21. P. M. PARDALOS AND J. XUE, *The maximum clique problem*, Journal of Global Optimization, 4 (1994), pp. 301–328.
22. M. PAVAN AND M. PELILLO, *A new graph-theoretic approach to clustering and segmentation*, in Proc. of the 2003 IEEE computer society conference on Computer vision and pattern recognition, CVPR'03, Washington, DC, USA, 2003, IEEE Computer Society, pp. 145–152.
23. S. SADI, S. ÖĞÜDÜCÜ, AND A. S. UYAR, *An efficient community detection method using parallel clique-finding ants*, in Proc. of IEEE Congress on Evol. Comp, July 2010, pp. 1–7.
24. P. SAN SEGUNDO, D. RODRÍGUEZ-LOSADA, AND A. JIMÉNEZ, *An exact bit-parallel algorithm for the maximum clique problem*, Comput. Oper. Res., 38 (2011), pp. 571–581.
25. E. TOMITA AND T. SEKI, *An efficient branch-and-bound algorithm for finding a maximum clique*, in Proc. of the 4th international conference on Discrete mathematics and theoretical computer science, Berlin, Heidelberg, 2003, Springer-Verlag, pp. 278–289.
26. J. TURNER, *Almost all k-colorable graphs are easy to color*, Journal of Algorithms, 9 (1988), pp. 63–82.

## Appendix: DIMACS Results

**Table 3.** Comparison of runtimes of algorithms [6] (*CP*) and [19] (*cliquer*), with that of our new exact algorithm ( $\tau_{new-exa}$ ) for DIMACS graphs. An asterisk (\*) indicates that the algorithm did not terminate within 10,000 seconds for that instance.  $\mu$  denotes the maximum clique size,  $\mu_{new-heu}$  the maximum clique size found by our heuristic and  $\tau_{new-heu}$ , its runtime.

$G$	$ V $	$ E $	$\mu$	$\tau_{CP}$	$\tau_{cliquer}$	$\tau_{new-exa}$	$\mu_{new-heu}$	$\tau_{new-heu}$
<i>brock200_1</i>	200	14,834	21	*	10.37	*	18	0.02
<i>brock200_2</i>	200	9,876	12	0.98	0.02	1.1	10	<0.01
<i>brock200_3</i>	200	12,048	15	14.09	0.16	14.86	12	<0.01
<i>brock200_4</i>	200	13,089	17	60.25	0.7	65.78	14	<0.01
<i>c-fat200-1</i>	200	1,534	12	<0.01	<0.01	<0.01	12	<0.01
<i>c-fat200-2</i>	200	3,235	24	<0.01	<0.01	<0.01	24	<0.01
<i>c-fat200-5</i>	200	8,473	58	0.6	0.33	0.93	58	0.04
<i>c-fat500-1</i>	500	4,459	14	<0.01	<0.01	<0.01	14	<0.01
<i>c-fat500-2</i>	500	9,139	26	0.02	<0.01	0.01	26	0.01
<i>c-fat500-5</i>	500	23,191	64	3.07	<0.01		64	0.11
<i>hamming6-2</i>	64	1,824	32	0.68	<0.01	0.33	32	<0.01
<i>hamming6-4</i>	64	704	4	<0.01	<0.01	<0.01	4	<0.01
<i>hamming8-2</i>	256	31,616	128	*	0.01	*	128	0.67
<i>hamming8-4</i>	256	20,864	16	*	<0.01	*	16	0.03
<i>hamming10-2</i>	1,024	518,656	512	*	0.31	*	512	95.24
<i>johnson8-2-4</i>	28	210	4	<0.01	<0.01	<0.01	4	<0.01
<i>johnson8-4-4</i>	70	1,855	14	0.19	<0.01	0.23	14	<0.01
<i>johnson16-2-4</i>	120	5,460	8	20.95	0.04	22.07	8	<0.01
<i>keller4</i>	171	9,435	11	22.19	0.15	23.35	11	<0.01
<i>MANN_a9</i>	45	918	16	1.73	<0.01	2.5	16	<0.01
<i>MANN_a27</i>	378	70,551	126	*	*	*	125	1.74
<i>p_hat300-1</i>	300	10,933	8	0.14	0.01	0.14	8	<0.01
<i>p_hat300-2</i>	300	21,928	25	831.52	0.32	854.59	24	0.03
<i>p_hat500-1</i>	500	31,569	9	2.38	0.07	2.44	9	0.02
<i>p_hat500-2</i>	500	62,946	36	*	159.96	*	34	0.14
<i>p_hat700-1</i>	700	60,999	11	12.7	0.12	12.73	9	0.04
<i>p_hat1000-1</i>	1,000	122,253	10	97.39	1.33	98.48	10	0.11
<i>san200_0.7_1</i>	200	13,930	30	*	0.99	*	16	0.01